"Validación de consistencia en la integración de datos a distintos niveles de granularidad"

Ana Catalina Retamal Pardo
Departamento de Ingeniería Informática
Universidad de Concepción

Nombre Profesora Guía: Andrea Rodríguez Tastets
Comisión: Guillermo Cabrera Vives, Lilian Salinas Ayala

4 de abril de 2018

Abstract

En el trabajo con bases de datos muchas veces se maneja información que se almacena a distintos niveles de granularidad. Los ejemplos más usuales de este tipo son información geográfica o temporal. Un problema común que se presenta al trabajar en estos dominios es la validación de la consistencia de datos cuando estos se encuentran a distintos niveles de granularidad, como es el caso, por ejemplo, de ventas por Ciudad y por Provincia, o por Día y por Año. En esta Memoria de Título se propone un algoritmo que permite verificar la consistencia de datos en estos casos considerando niveles de redondeo distintos para cada granularidad, respecto a restricciones de integridad que condicionan los valores de agregación de atributos. Para ello se trabaja con un caso específico, las votaciones en la segunda vuelta de las elecciones presidenciales de Chile del año 2013, donde se dispone de datos almacenados para distintas divisiones administrativas y electorales. A partir de este caso, se desarrolló un algoritmo que generaliza a otros casos similares.

1. Introducción

Las bases de datos son utilizadas actualmente en diversos ámbitos, y en ellas se maneja información de todo tipo. En este contexto, es común que se requiera integrar o comparar datos provenientes de distintos orígenes o que tienen diferentes niveles de granularidad o de redondeo, y en estos casos es importante asegurarse de que los datos sean consistentes entre sí. El trabajo realizado en esta Memoria de Título tiene relación con esto último.

Entenderemos que en las bases de datos existen *atributos granulares*, es decir, que pueden ser representados a distintos niveles de granularidad, y que generalmente estos corresponden a atributos espaciales o temporales. Por ejemplo, considerar dos bases de datos que almacenan la población, una a nivel de país (en millones) y la otra a nivel de regiones de Chile (en miles):

Tabla 1: **Población por país**

País	**Población (millones)**
Chile	17,91
Argentina	43,85
Brasil	207,7

Tabla 2: **Población por región**

Región	**Población (miles)**
Metropolitana	7112
Bíobío	2037
Valparaíso	1815

Si bien los datos en ambas tablas representan una misma variable, la población, estos se encuentran a niveles de granularidad distintos: La Tabla 1 muestra los datos a nivel de país y redondeados al millón, mientras que la Tabla 2 muestra los datos a nivel de región y redondeados a miles.
Se esperaría que los datos contenidos en estas dos tablas sean consistentes entre sí. Es decir, que al agregar (sumar) los datos correspondientes a todas las regiones que conforman Chile, se obtenga un resultado parecido a la población de Chile en la tabla de países.

$$A_1 + A_2 + ... + A_n \approx B_i$$

Este caso es un ejemplo de que deben existir **reglas** que se cumplan al realizar el chequeo de consistencia.

Este trabajo está estrechamente relacionado con *A Model for Multigranular Data and its Integrity (S. Hegner & M. A. Rodrguez, 2016)*, donde se presenta una formalización de los modelos multigranulares como el mencionado anteriormente, y se entregan definiciones de

las restricciones de integridad como extensión a dependencias funcionales de relaciones de orden. Lo que se propone en esta Memoria de Título es una implementación de dicho modelo teórico orientado a la evaluación de consistencia.

1.1. Objetivos

1.1.1. Objetivo General

Diseñar e implementar un modelo de datos que permita verificar la consistencia de la información entregada por bases de datos a diferentes niveles de granularidad.

1.1.2. Objetivos Específicos

- Estudiar el modelo de datos presentado en [1].
- Diseñar el modelo de datos necesario para representar el esquema multigranular.
- Desarrollar un algoritmo capaz de comprobar la consistencia de los datos a distintos niveles de granularidad.
- Implementar el sistema de acuerdo a lo establecido en las etapas anteriores.

1.2. Metodología de Trabajo

El trabajo realizado consistió en diseñar una base de datos que contiene la estructura de gránulos y granularidades, así como el algoritmo que, haciendo uso de las restricciones de integridad y el conocimiento de los gránulos y granularidades, consulte a diferentes bases de datos y verifique si la información contenida es consistente entre estas.
Los componentes generales del sistema implementado se muestran en la Figura 1. El sistema consta de varias bases de datos BD_i que tienen **un mismo esquema**, pero que almacenan la información a **distintos niveles de granularidad**. Además se implementó otra base de datos que almacena el esquema multigranular que representa el conocimiento del dominio, y el algoritmo utilizará esta información en conjunto con las TMCD (*Thematic Multigranular Comparision Dependencies*) para verificar la consistencia de datos.

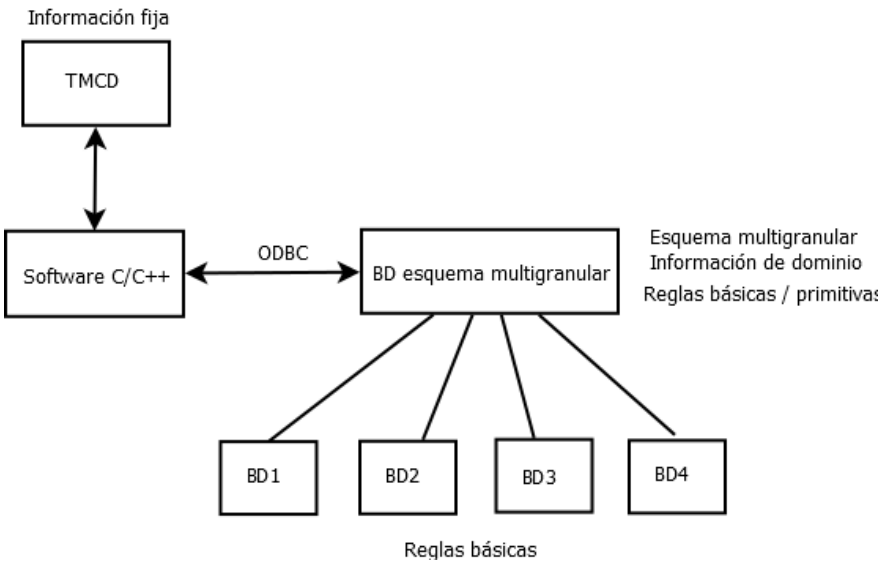

Figura 1: Componentes del sistema implementado

- Las bases de datos se implementaron utilizando **Postgres**, que es un motor de bases de datos open source.
- El algoritmo de verificación de consistencia fue implementado en PL/pgSQL, utilizando el estándar de acceso a bases de datos *Open DataBase Connectivity* (ODBC).
- Una vez implementado el algoritmo, se poblaron bases de datos con esquemas iguales pero con valores a distintos niveles de granularidad y tanto con datos consistentes como inconsistentes, estudiando el comportamiento del programa para los distintos casos con respecto al tiempo de ejecución y resultados entregados.

1.3. Organización del Documento

El contenido de este documento en las secciones siguientes está organizado en cinco capítulos.
En el capítulo "Discusión Bibliográfica" se describen brevemente dos trabajos relacionados con el modelo y el algoritmo desarrollados para esta Memoria de Título, *Un Modelo para Datos Multigranulares y su Integridad (S. Hegner & M. A. Rodríguez, 2016)* y *Esquemas Multi-Granulares para Integración de Datos (M. A. Rodríguez & L. Bravo, 2012)*

Posteriormente se encuentra el capítulo "Modelo de Datos y Algoritmo Desarrollado", que está dividido en tres secciones: La primera, "Modelo de Datos", contiene definiciones de conceptos y supuestos que fueron aplicados en la realización de este trabajo. Luego, la secci ón "Implementación de un Modelo Multigranular" describe todo el trabajo realizado para adaptar la base de datos en particular con la que se trabajó al esquema multigranular,

describiendo las tablas y los algoritmos que se utilizaron para poblarlas correctamente. La tercera sección de este capítulo, llamada "Desarrollo del Algoritmo de Chequeo de Consistencia", explica cómo se implementó el algoritmo de chequeo de consistencia. Le sigue la sección "Determinación de un Umbral de Error", donde se mencionan algunos factores que debieran influir en los resultados obtenidos, y qué se debiera tener en cuenta al fijar un valor de tolerancia o umbral Tol para el algoritmo.

El capítulo siguiente es "Experimentos y Resultados", que está dividido en dos secciones: Experimentos relacionados con tiempo de ejecución y experimentos relacionados con el valor de la variable $avgdif$, variable cuyo comportamiento determina qué valores de tolerancia son efectivos para el correcto chequeo de consistencia.

Por último, en el capítulo "Conclusión" se hace un resumen del trabajo hecho y se mencionan algunas direcciones en las que pudiera complementarse el trabajo realizado.

2. Discusión Bibliográfica

2.1. Un Modelo para Datos Multigranulares y su Integridad

El trabajo de mayor influencia para la realización de esta Memoria de Título es *A Model for Multigranular Data and its Integrity (S. Hegner & M. A. Rodríguez, 2016)* [1]. En él se desarrolla un modelo de datos multigranulares que soporta no sólo la estructura usual de orden de granularidades, sino que también relaciones de retículo (*lattice*) con operaciones de unión y unión disjunta, de manera que los gránulos pueden relacionarse entre sí de maneras mucho más complejas.
El artículo mencionado propone como ejemplo un caso similar al descrito anteriormente en este documento: Se tienen dos bases de datos BD_1 y BD_2 con esquemas iguales, pero datos almacenados a granularidades diferentes.

Tabla 3: **Base de Datos 1**

Lugar	**Tiempo**	**Valor**
Región I	Q1Y2014	n_1
Región II	Q1Y2014	n_2
...	...	...
Región XV	Q1Y2014	n_15

Tabla 4: **Base de Datos 2**

Lugar	**Tiempo**	**Valor**
Chile	Q1Y2014	b_1
Chile	Q2Y2014	b_2
Chile	Q3Y2014	b_3
Chile	Q4Y2014	b_4

Cuando se quiere establecer si existe consistencia de datos en este tipo de casos, la restricción de integridad más importante entre estas tablas debe ser que dada una fila $\langle Lugar,\ Tiempo,\ n_i\rangle$ existente en BD_1 y una fila $\langle Lugar,\ Tiempo,\ b_j\rangle$ en BD_2, siempre que las variables *Lugar* y *Tiempo* tengan los mismos valores en ambas tablas, se debe cumplir que $n_i = b_j$.
Sin embargo, existen más reglas que se deben satisfacer. Por ejemplo, sabemos que existen atributos de tipo espacio-temporal que están contenidos dentro de otros, como es el caso de las provincias que están contenidas dentro de regiones, o de los trimestres de un año (*Quarter-Year*) que están contenidos dentro de él.

$$Concepcion \sqsubseteq Biobio \ (1) \qquad Q2Y2014 \sqsubseteq Y2014 \ (2)$$

Si tuviera dos tuplas $\langle Concepcion,\ Q2Y2014,\ n_C\rangle$ y $\langle Biobio,\ Y2014,\ n_B\rangle$, y considerando (1) y (2), debiera cumplirse que $n_C \leq n_B$. En este caso se puede entender como gránulo al

área espacial o fracción de tiempo en sí, es decir, a Concepción o a Q2Y2014, mientras que se llama granularidad al nivel de detalle al que se encuentran los gránulos. Por ejemplo, la granularidad de Concepción es Ciudad y la de Q2Y2014 es Cuarto de Año.

El trabajo de S. Hegner y M. A. Rodríguez plantea una formalización de los esquemas de datos multigranulares como éste, utilizando un enfoque en que se definen primero las estructuras de las granularidades y gránulos, y después la semántica de las reglas a través de la satisfacción del modelo de datos. En dicha formalización se propone que las relaciones existentes entre granularidades de un esquema forman un conjunto parcialmente ordenado o *poset*, donde algunas granularidades (pero no necesariamente todas) pueden relacionarse entre sí de la manera $Glty_1 \sqsubseteq Glty_2$, donde $\sqsubseteq$ representa relaciones de contención, es decir, que todos los gránulos con la granularidad $Glty_1$ están completamente contenidos dentro de un único gránulo de granularidad $Glty_2$. En el caso de las divisiones administrativas de Chile esto permite plantear relaciones como Provincia $\sqsubseteq$ Región, y a partir de esto derivar reglas pertinentes a los gránulos en sí, llamadas *Asignaciones de Gránulo*, que relacionan cada gránulo con su correspondiente granularidad. Además se explicitan restricciones propias de la estructura multigranular, como el hecho de que todos los gránulos a un mismo nivel de granularidad no pueden estar superpuestas entre sí (su intersección es vacía).

El modelo propuesto considera todas las restricciones planteadas para proponer una manera de realizar verificaciones de integridad de información en distintas bases de datos, a través del cumplimiento de *Thematic Multigranular Comparision Dependencies* (TMCD) (ver Tabla 5). Estas reglas son extensiones a las clásicas dependencias funcionales que tienen la forma $[A_1, A_2, \ldots, A_k]_{\langle\beta,\eta\rangle} \longrightarrow \langle B, \oplus\rangle$, donde:

- t_1 y t_2 son tablas cuyas filas tienen atributos granulares y atributos temáticos. Para las TMCD mostradas en la tabla 5, se asume que los elementos de la tabla t_2 pueden agregarse de manera que la unión puede compararse con un elemento de la tabla t_1.
- A_i son atributos granulares del esquema, por ejemplo regiones o provincias.
- B es el *Atributo Temático* cuyo valor depende funcionalmente de los demás atributos y sobre el que se hace la agregación. Ejemplos de atributos de este tipo pueden ser la población asociada a una región o provincia.
- β es un parámetro que representa el tipo de *join* que se considera entre los gránulos y puede tomar dos valores: $\bigsqcup\!\!\!\bigsqcup$ (*join disjunto*), que significa que los gránulos A_i no se intersectan entre sí; o $\bigsqcup$ (*join simple*), que significa que la intersección entre los elementos A_i puede ser no nula. En cada caso, las TMCD asociadas serán distintas.
- η es el tipo de *orden* y representa la relación existente entre los gránulos agregados de la tabla t_2 y un gránulo de la tabla t_1. Esta relación puede ser de tipo $\sqsubseteq$, $=$ ó $\sqsupseteq$. r $\sqsubseteq$, $=$ ó $\sqsupseteq$

De este modo, considerando todas las combinaciones posibles de valores de β y η, se pueden diferenciar seis tipos de TMCD, que se muestran en la tabla 5.

Tabla 5: TMCD para distintos valores de β y η

Tipo ($\langle\beta,\ \eta\rangle$)	**Regla de Comparación**	**Agregación**
$\langle \bigsqcup : = \rangle$	$t_1.A_i = \bigsqcup t_2.A_i$	$t_1.B = \bigoplus t_2.B$
$\langle \bigsqcup : \sqsubseteq \rangle$	$t_1.A_i \sqsubseteq \bigsqcup t_2.A_i$	$t_1.B \leq \bigoplus t_2.B$
$\langle \bigsqcup : \sqsupseteq \rangle$	$t_1.A_i \sqsupseteq \bigsqcup t_2.A_i$	$t_1.B \geq \bigoplus t_2.B$
$\langle \bigsqcup : = \rangle$	$t_1.A_i = \bigsqcup\ t_2.A_i$	$t_1.B = \bigoplus t_2.B$
$\langle \bigsqcup : \sqsubseteq \rangle$	$t_1.A_i \sqsubseteq \bigsqcup\ t_2.A_i$	$t_1.B \leq \bigoplus t_2.B$
$\langle \bigsqcup : \sqsupseteq \rangle$	$t_1.A_i \sqsupseteq \bigsqcup\ t_2.A_i$	$t_1.B \geq \bigoplus t_2.B$

Lo anterior puede ilustrarse más claramente con un ejemplo. Consideremos dos tablas de datos t_a y t_b, cuya estructura y contenidos son tales como los que se muestran en las tablas 6 y 7.

Tabla 6: **Habitantes por Ciudad** (t_a)

Gránulo	**Granularidad**	**Habitantes**
Concepción	Ciudad	n_1
Chillán	Ciudad	n_2

Tabla 7: **Habitantes por Región** (t_b)

Gránulo	**Granularidad**	**Habitantes**
Biobío	Región	m_1

En la tabla de TMCD, t representa las filas de cada tabla, mientras los valores A_i representan los gránulos que en este caso corresponden a Concepción, Chillán y Biobío. B son los valores de los *atributos temáticos*, es decir, los que sufren agregación, y que en este ejemplo corresponden a n_1, n_2 y m_1. Para verificar la consistencia entre los datos contenidos en estas dos tablas es necesario distinguir a qué caso particular de la tabla de TMCD corresponde el ejemplo, para así construir la regla de chequeo de consistencia correcta. Esto se hace considerando dos factores:

1. Si la intersección entre los gránulos de la tabla con la granularidad más fina es nula, el valor de β es $\bigsqcup$. De lo contrario, su valor es $\bigsqcup$.

2. Si la unión (sea ésta $\bigsqcup$ ó $\bigsqcup$) de los gránulos de tabla con la granularidad más fina es igual al gránulo con que se comparará la consistencia en la otra tabla, el valor de η es "$=$". Si esta unión es menor, η es igual a "$\sqsubseteq$", y si es mayor, su valor es "$\sqsupseteq$".

Continuando con el ejemplo, sabemos que Chillán y Concepción no se intersectan (puesto que no hay habitantes que vivan en ambas ciudades al mismo tiempo), y por lo tanto el parámetro β corresponde a $\sqcup$. Además, sabemos que al sumar los habitantes de Chillán y Concepción no se tiene a todos los habitantes de la Región del Biobío, puesto que hay personas que viven en la región pero no viven en ninguna de estas ciudades. Por lo tanto, la unión de Chillán y Concepción es menor a la Región del Biobío, y el valor de η es "$\sqsupseteq$". Luego, según la Tabla 5, la regla de chequeo de consistencia para este caso debe ser de la siguiente:

$$t_b.n_j \geq \sum_{i=1}^{2} t_a.m_i$$

En el ejemplo anterior el operador de agregación ($\bigoplus$) es la suma, por lo tanto se reemplaza por el símbolo de la sumatoria en la regla final. El paper señala que también se puede trabajar con otros operadores de agregación, siempre y cuando cumplan con las propiedades de monotonicidad y duplicado-invariancia.

Por último, el trabajo de Hegner y Rodríguez afirma que es apropiado considerar un *umbral* de error en las comparaciones. Sin embargo, los autores no establecen una manera de determinar dicho valor, puesto que va más allá de los objetivos del artículo.

El modelo multigranular basado en reglas propuesto en este trabajo no es el único existente para este tipo de casos. Se puede utilizar también un modelo donde se represente la geometría de los objetos en un plano $\mathbb{R}^2$ o el espacio $\mathbb{R}^3$ para cada gránulo. Si bien este tipo de modelos puede almacenar toda la información que pudiera ser relevante para la gran mayoría de las consultas, en la práctica resulta demasiado costoso tanto en espacio como tiempo utilizado para consultas o mantenimiento. Para ejemplificar esto, en la mayoría de las aplicaciones de bases de datos no es necesario saber cuáles son las coordenadas exactas donde se encuentra cada región de Chile, únicamente basta con saber que una región está compuesta por tales comunas. El modelo de reglas propuesto por S. Hegner y M. A. Rodríguez ofrece la ventaja de tener menores costos de almacenamiento y de consultas, manteniendo a la vez la complejidad de las relaciones entre gránulos y granularidades.

2.2. Esquemas Multi-Granulares para Integración de Datos

El trabajo anterior está relacionado a su vez con *Multi-Granular Schemas for Data Integration (M. A. Rodríguez & L. Bravo, 2012)*. En este artículo se propone un esquema multigranular de dominio que puede ser usado en la formalización de esquemas de bases de datos, de manera que a cada atributo se le asigna un nivel de granularidad.

Este trabajo es similar al de S. Hegner y M. A. Rodríguez, pues se plantea también una manera de representar los esquemas de datos multigranulares, almacenando también información referente al esquema de las granularidades en sí y mostrando cómo utilizar estos

datos sobre los datos para facilitar operaciones de integración entre distintos esquemas.
Para la integración de datos a distintos esquemas se plantea la idea de encontrar un esquema global que permita comparar los datos contenidos en ambos esquemas. Por ejemplo, si se tuviera una base de datos a nivel de Mes y otra a nivel de Año, la idea es buscar un esquema que permita comparar ambas bases. Se puede convertir la base de datos a nivel Mes a una nueva base de datos a nivel Año, lo que permitiría realizar las comparaciones. Sin embargo, se distinguen también casos donde el esquema global no es uno de los dos esquemas, por lo que hay que convertir las dos bases de datos a un tercer esquema. En este caso se habla del *dominio de integración más fino*, que permita realizar las comparaciones con el mayor nivel de precisión posible. Por ejemplo, si tengo datos asociados a Mes y datos asociados a Semana, y puesto que existen semanas que no están totalmente contenidas dentro de un mes, no es posible convertir estos últimos datos al primer esquema. Es necesario encontrar un tercer esquema que englobe a los dos y permita trabajar con los datos de ambas bases.
Para realizar la tarea de la conversión de esquemas, el artículo propone un algoritmo *MergedSchema* que permite encontrar este esquema global. El algoritmo propuesto es similar a la rutina de Conversión entre Granularidades *gltyconv* desarrollada en el trabajo presente. Sin embargo, y puesto que el trabajo de M. A. Rodríguez y L. Bravo no pone énfasis en ningún esquema de datos en particular, la rutina está planteada sólo en términos generales como pseudocódigo y no fue implementada con datos reales.

3. Modelo de Datos y Algoritmo Desarrollado

3.1. Modelo de Datos

En esta sección se describirá el modelo de datos multigranular utilizado para trabajar, así como la notación utilizada y algunos conceptos previos sobre los que se realizó el trabajo. Gran parte del modelo multigranular de datos se desarrolló en *A Model for Multigranular Data and its Integrity (S. Hegner & M. A. Rodríguez, 2016).*

3.1.1. Conceptos previos

- **Granularidad**
  Se entenderá por granularidad el nivel de detalle en que se almacenan los datos en una tabla. Generalmente los atributos granulares son de índole espacio-temporal.
  Algunos ejemplos de granularidades espaciales son ciudad, provincia, región y país; mientras que algunos ejemplos de granularidades temporales son hora, día, mes y año.

- **Gránulo**
  Entenderemos por gránulo a cualquier valor de atributo que se encuentre a un nivel de granularidad. A continuación se muestran algunos ejemplos de gránulos con su respectiva granularidad. En la Tabla 8 se muestran gránulos con granularidades de tipo espacial, mientras que en la Tabla 9 se muestran gránulos con granularidades de tipo temporal.

Tabla 8: Gránulos con granularidades de tipo espacial

Gránulo	**Glty**
Antofagasta	Comuna
III Región	Región
Chile	País

Tabla 9: Gránulos con granularidades de tipo temporal

Gránulo	**Glty**
15/06/2011	Día
Noviembre 2013	Mes
2013	Año

- **Estructura de Orden**
  Al tener varios gránulos de un mismo tipo (espacial o temporal), se puede establecer entre ellos una relación de contención o subsunción cuyo símbolo es " $\sqsubseteq$ ".
  Esta relación representa la inclusión de un gránulo dentro de otro, es decir, si g_1 está contenido completamente dentro de g_2, podremos decir que $g_1 \sqsubseteq g_2$.

$$\text{Temuco} \sqsubseteq \text{IX Región} \qquad 03/08/2012 \sqsubseteq \text{Año 2012}$$

Es importante observar que si un gránulo g_a está sólo parcialmente contenido dentro de un gránulo g_b, no podremos decir que g_b subsume a g_a. Por ejemplo, la quinta semana del año 2018 empieza el 29 de Enero y termina el 4 de Febrero, y no se encuentra contenida totalmente dentro del mes de Enero ni del mes de Febrero.

$$\text{Semana 5 del 2018} \not\sqsubseteq \text{Enero 2018} \qquad \text{Semana 5 del 2018} \not\sqsubseteq \text{Febrero 2018}$$

La inclusión de este operador " $\sqsubseteq$ " al conjunto de gránulos forma un **orden parcial** y, por lo tanto, constituye un **conjunto parcialmente ordenado** o **poset**.
Esto le confiere propiedades importantes al conjunto, que serán de utilidad para el diseño del algoritmo.

- **Orden parcial de Granularidades**
  A partir de la estructura de orden parcial de gránulos descrita en el punto anterior, podemos establecer a su vez un orden parcial entre las granularidades.
  Diremos que una granularidad $Glty_1$ está contenida dentro de $Glty_2$ si para todos los gránulos g_1 de granularidad $Glty_1$ existe un gránulo g_2 tal que $g_1 \sqsubseteq g_2$.

$$\forall g_1 \in Glty_1 \; \exists g_2 \in Glty_2 \; : \; g_1 \sqsubseteq g_2 \Rightarrow Glty_1 \sqsubseteq Glty_2$$

  De esta manera, podemos decir que algunas relaciones de contención o subsunción existentes entre las granularidades representadas por las divisiones administrativas de Chile son las siguientes:

$$\text{Comuna} \sqsubseteq \text{Provincia} \qquad \text{Comuna} \sqsubseteq \text{País}$$

  Sin embargo, al igual que en el caso anterior de los gránulos, es posible que existan pares de granularidades que sean no comparables, es decir, donde no puede decirse que una de las dos es menor o igual que la otra. Esto es lo que diferencia la estructura de orden parcial de un orden total, donde todos los pares de elementos son comparables.

- **Relación de Cobertura**
  La relación de cobertura de un orden parcial asocia a dos elementos de este orden sólo si estos son vecinos inmediatos. Diremos que a cubre a b si $a \sqsubset b$, pero no existe un elemento c tal que $a \sqsubset c \sqsubset b$. Esta relación la representaremos como $a \sqsubset\!\cdot\, b$

$$a \sqsubset b \; \wedge \nexists \, c : a \sqsubset c \sqsubset b \Rightarrow a \sqsubset\!\cdot\, b$$

  Del mismo modo, hablaremos de que dos granularidades son adyacentes si existe entre ellas una relación de cobertura.
  Como ejemplo de esta adyacencia de granularidades se puede estudiar el caso de las divisiones administrativas de Chile. Suponiendo que tenemos un conjunto de granularidades que incluye comuna, provincia y región, las relaciones de cobertura existentes serían las siguientes:

Comuna $\sqsubset\!\cdot$ Provincia Provincia $\sqsubset\!\cdot$ Región

- **Supremo e Ínfimo**
  En un conjunto parcialmente ordenado $(K, \leq)$, dado un subconjunto $C \subset K$, se define como *cota superior* de C a un elemento de K que es mayor o igual que todos los elementos en C. La menor de las cotas superiores de C se conoce como supremo (*Least Upper Bound*, abreviada como LUB) de C.
  Análogamente, se define la *cota inferior* de C como un elemento de K que es menor o igual que todos los elementos de C. La mayor de las cotas inferiores de C se conoce como ínfimo (*Greatest Lower Bound*, abreviada como GLB) de C.

- **Restricciones para la Integridad de Datos**
  La evaluación de consistencia de datos que se realizará en el algoritmo fue desarrollada en base a las restricciones de integridad propuestas en el trabajo de S. Hegner y M. A. Rodríguez (2016). El algoritmo verifica el cumplimiento de las *Thematic Multigranular Comparision Dependencies* (TMCD), que fueron descritas en la Discisión Bibliográfica. En particular, el alcance de este trabajo sólo cubrirá las TMCD del primer caso mostrado en la Tabla 5, es decir, la verificación de consistencia de datos cuando el tipo de join (β) es "$\sqcup$" y el tipo de orden (η) es "=". Además, la operación de agregación ($\bigoplus$) para el caso particular escogido es la suma. En otras palabras, la verificación de consistencia da por supuesto que:

  1. Los gránulos de la tabla a agregar tienen una intersección nula entre sí.
  2. Los gránulos de la tabla a agregar pueden unirse de manera que la unión resultante es siempre **igual** a uno o más gránulos de la granularidad a agregar.

  Más adelante, en el capítulo de Implementación de un Modelo Multigranular, se verá que el caso de ejemplo escogido cumple con estas propiedades.

  De esta manera, la regla de consistencia a verificar es

  $$t_1.B = \bigoplus t_2.B$$

  Donde t_1 es una fila (gránulo) de la tabla con la granularidad más grande y t_2 son las filas (gránulos) de la tabla con granularidad menor que unidas forman al elemento t_1. Como la unión de los elementos de t_2 es igual al gránulo de t_1 (supuesto 2) y además los gránulos de t_2 no se intersectan entre sí, su suma debe ser igual al valor B del elemento en t_1.

- **Redondeo**
  Se definirá el *redondeo* de un número como una función $rounding(x,\ y) \longrightarrow z$ que dados dos valores x e y, entrega un resultado z que es el múltiplo de y que es *más cercano* al valor de x. Los casos más comúnes de la función de redondeo son los que tienen un valor y correspondiente a una potencia de 10, de los cuales se muestran algunos ejemplos.

$$rounding(55,\ 10) = 60 \qquad\qquad rounding(1841,\ 1000) = 2000$$

Sin embargo, la definición que se utiliza en este trabajo permite trabajar también con valores de y que no son potencias de 10. Dicha función se entenderá de la siguiente manera:

$$rounding(x,\ y) = \begin{cases} x & si\ (x \mod y) \leq \frac{y}{2} \\ \lfloor \frac{x}{y} \rfloor * y + y & si\ (x \mod y) > \frac{y}{2} \end{cases}$$

Donde mód es el operador módulo que entrega el resto resultante de la división entre dos números, y $\lfloor a \rfloor$ es la función piso de a, que entrega el mayor número entero que es menor o igual a a.
Puede verse que los redondeos usuales de potencias de 10 son casos particulares de la función definida, donde el valor de y equivale a dicha potencia de 10. Esta función de redondeo permite extender el concepto de redondeo a cualquier entero positivo.

Si bien existen numerosas definiciones de la función redondeo, la enunciada anteriormente será la que se utilizará para este trabajo. Cuando se redondee un número x, teniendo que $rounding(x,\ y) = z$, se hablará de que z es igual a x *redondeado a* y, o bien, que el nivel de redondeo de z es y.

3.2. Implementación de un Modelo Multigranular

La primera etapa del proyecto de memoria de título consistió en la implementación de una base de datos con una estructura multigranular, que pueda ser usada como ejemplo para aplicar los algoritmos de chequeo de consistencia. En particular, los datos que se escogieron como ejemplo para ser implementada con dicha estructura fueron los correspondientes a las Votaciones de la Segunda Vuelta de las Elecciones Presidenciales de Chile del año 2013, en la que fueron candidatos a la presidencia Michelle Bachelet y Evelyn Matthei. El Servicio Electoral de Chile (SERVEL) publicó estos datos tras la elección a distintos niveles de granularidad. Por un lado, se publicaron las votaciones por Mesa Electoral, y por otro lado se entregó esta información agrupada por Regiones de Chile. Además de estas dos instancias, Chile cuenta con distintas divisiones administrativas y electorales con distintas relaciones entre sí, por lo que resulta una instancia interesante en la que se pueden analizar distintos casos de agrupación y refinamiento de datos para el posterior chequeo de consistencia entre ellos.

3.2.1. Divisiones Administrativas y Electorales de Chile

El territorio chileno se suele dividir de dos maneras importantes cuando se trata de estudiar resultados electorales:

- En primer lugar existen las divisiones administrativas, que son las que determinan la manera en que se organiza el país. De acuerdo a esta organización territorial, para el año 2013 el país se dividía en 16 Regiones, que se dividen en 56 Provincias y éstas a su vez se dividían en 346 Comunas.

$$\text{Comuna} \sqsubseteq \text{Provincia} \sqsubseteq \text{Región}$$

- Además existen las divisiones electorales, que organizan el territorio con fines de elecciones parlamentarias y de consejeros regionales. De acuerdo a esta repartición territorial, y desde el año 1998, el país puede dividirse en 19 Circunscripciones Senatoriales, que se dividen en 60 Distritos, divididos a su vez en 346 Comunas.

$$\text{Comuna} \sqsubseteq \text{Distrito} \sqsubseteq \text{Circunscripción Senatorial}$$

Sin embargo, para fines de conteo y organización de votos, el Servicio Electoral de Chile divide cada una de las 346 Comunas en 572 Circunscripciones Electorales, y éstas están compuestas en total por 41349 Mesas Electorales.

$$\text{Mesa Electoral} \sqsubseteq \text{Circunscripción Electoral} \sqsubseteq \text{Comuna}$$

Un detalle importante a considerar es que, pese a que en teoría el país consta de 346 comunas, actualmente la comuna de la Antártica Chilena carece de una municipalidad propia, por lo que el SERVEL entrega sus datos agrupados dentro de la comuna de Cabo de Hornos. Es por este motivo que las bases de datos publicadas por este organismo contienen 345 comunas.

3.2.2. Cumplimiento de las propiedades de join y orden

El dominio granular escogido cumple con las dos propiedades que se describieron en el capítulo anterior.

1. Las divisiones administrativas de un mismo nivel no se intersectan, puesto que no existen votos que pertenezcan a más de una mesa electoral, comuna, distrito, etc. al mismo tiempo. Por lo tanto, **siempre y cuando todos los elementos de una tabla sean de la misma granularidad**, la intersección entre ellos es nula.

2. Para cualquier granularidad $Glty_1$ que esté contenida dentro de una granularidad $Glty_2$, se cumple que para todos los gránulos $g_1 \in Glty_1$, la unión de los gránulos $g_2 \in Glty_2$ con los que tiene una relación de cobertura $g_2 \sqsubset\!\cdot\, g_1$ es igual a g_2. Esto quiere decir, por ejemplo, que la unión de todas las provincias que conforman una región son iguales a esa región, y que no existen votos que pertenezcan a una región pero no pertenezcan a ninguna comuna contenida dentro de esa región.

Esto quiere decir que la verificación de consistencia se hará mediante las TMCD del primer caso de la tabla 5 ($\langle \bigsqcup, = \rangle$).

3.2.3. Estructura de la Base de Datos

La base de datos se estructuró con el objetivo de poder representar de manera sencilla la información almacenada a distintos niveles de granularidad, así como las relaciones que existen entre estas granularidades. Todo esto con el fin de poder realizar los cálculos que sean necesarios y al mismo tiempo evitar almacenar información redundante.

En la Figura 2 se muestra el modelo entidad relación (MER) asociado a la base de datos implementada. El modelo consta de cuatro entidades: gránulos, granularidades, votos y candidatos. Cada voto está asociado a un candidato presidencial y a un gránulo, que corresponde a la división territorial donde éste fue emitido. Los gránulos, a su vez pertenecen a una determinada granularidad, y pueden relacionarse con otros gránulos por medio de relaciones de cobertura. Por su parte, las granularidades también pueden relacionarse con otras granularidades formando un orden. El modelo descrito fue posteriormente convertido a modelo relacional, con el que se trabajó finalmente. Las tablas resultantes se describen a continuación.

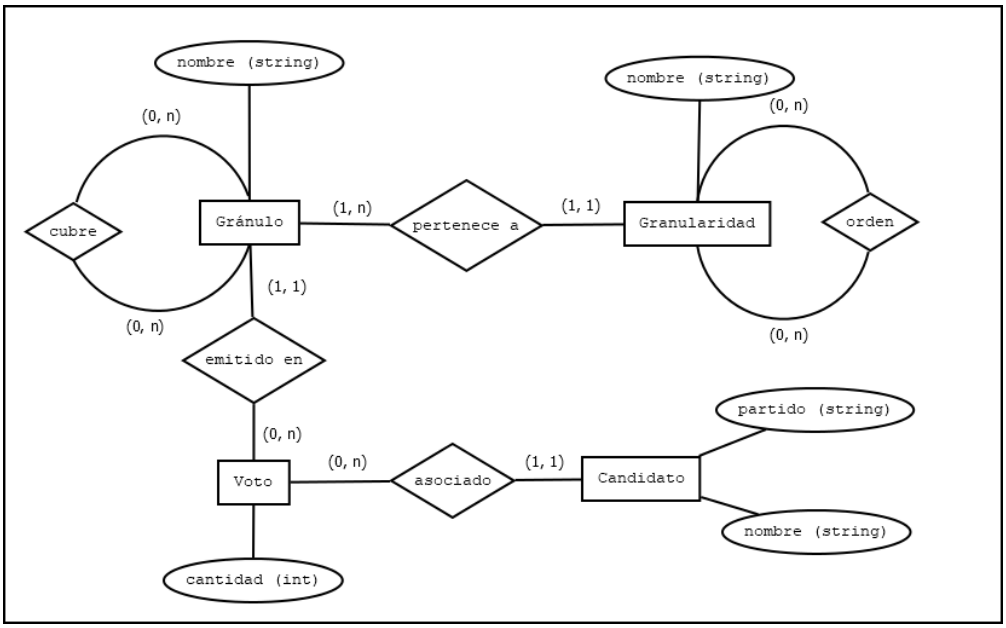

Figura 2: Modelo Entidad-Relación de la base de datos

El esquema relacional de la base de datos consta de dos tipos de tablas: Las tablas que describen el dominio granular y las tablas de instancia.

Tablas del Dominio Granular

Estas tablas son las que describen las granularidades y gránulos del caso de estudio, así como las relaciones existentes entre estos elementos. Se diferencian de las tablas de instancia porque son *fijas* para este caso de estudio, ya que corresponden a las tablas que se almacenan en la base de datos de esquema multigranular mostrados en la Figura 1.

- **gtty (name)**
  Esta tabla contiene las granularidades a las que se almacenan los datos (Por ejemplo Región, Provincia, Distrito, etc). Tiene una única columna *name* que es clave primaria y contiene el nombre de la granularidad.

- **granule (name, granularity)**
  Contiene los gránulos (en este caso las divisiones territoriales). Por ejemplo, en esta tabla se almacenan la Región del Bíobío, la Provincia de Concepción, la Mesa Electoral 24 M, etc. Contiene una columna *name* con el nombre de la granularidad y una columna *granularity* con su granularidad. Debido a que existen gránulos que tienen un mismo nombre pero se encuentran a distintas granularidades, esta tabla utiliza una clave primaria compuesta *(name, granularity)*.
  El hecho de que los gránulos estén almacenados todos en una misma tabla es algo que

se puede hacer para este caso producto del tamaño del dominio, pero es posible que en otros casos sea necesario separar los gránulos en distintas tablas.

- **covers (name1, gtty1, name2, gtty2)**
  Esta tabla almacena la relación de cobertura ($\sqsubset\!\cdot$) entre dos gránulos. Por ejemplo, esta tabla contiene el hecho de que la Comuna de Antofagasta está dentro de la Provincia de Antofagasta.
  Las dos primeras columnas, *name1* y *gtty1* hacen referencia a un gránulo g_1 de la tabla *granule* que es subsumido por el gránulo g_2, también de la tabla *granule*, al que hacen referencia las columnas *name2* y *gtty2*.

- **gttyorder (gtty1, gtty2)**
  De manera similar a la tabla anterior, esta tabla contiene relaciónes de cobertura, pero esta vez entre granularidades.
  Por ejemplo, esta tabla contiene la fila (Comuna, Provincia) que indica que todas las provincias pueden formarse por la unión de ciertas comunas.
  Contiene una columna *gtty1* que hace referencia a una granularidad $Glty_1$ de la tabla *granularity* y que es subsumida por una granularidad $Glty_2$, representada por la columna *gtty2* que también hace referencia a la tabla *granularity*

- **joinrel (id, name, gtty)**
  Esta tabla almacena cada gránulo que se puede formar uniendo gránulos de otra granularidad. Funciona en conjunto con la tabla glist para poder obtener todos los gránulos que unidos conforman a un gránulo de otra granularidad.
  Contiene una columna *id* que es un identificador numérico de la fila, y dos columnas *name* y *gtty* que en conjunto hacen referencia a un gránulo de la tabla *granule*.

- **glist (id, name, gtty)**
  Almacena la lista de gránulos que unidos conforman un gránulo de la tabla joinrel. Contiene una columna *id* que es una referencia a una fila de la tabla *joinrel*, y dos columnas *name* y *gtty* que en conjunto apuntan a un elemento de *granule*.

Por ejemplo, supongamos que tengo las tablas *joinrel* y *glist* que se muestran a continuación:

Tabla 10: Datos de ejemplo en la tabla *joinrel*

joinrel

id	**name**	**gtty**
8	Biobío	Región
9	Araucanía	Región
10	Los Lagos	Región

Tabla 11: Datos de ejemplo en la tabla *glist*

glist

id	**name**	**gtty**
8	Biobío	Provincia
9	Malleco	Provincia
9	Cautín	Provincia
10	Osorno	Provincia
10	Llanquihue	Provincia

Si quisiera conocer las provincias que conforman la Región de la Araucanía, bastaría con seleccionar el gránulo correspondiente en la tabla *joinrel* y hacer una operación *join natural* ($\bowtie$) con la tabla *glist*, uniéndose ambas tablas en la columna *id*. El resultado sería el siguiente:

Tabla 12: Resultado de la operación join natural entre las tablas 10 y 11

id	**joinrel.name**	**joinrel.gtty**	**glist.name**	**glist.gtty**
9	Araucanía	Región	Malleco	Provincia
9	Araucanía	Región	Cautín	Provincia

En conjunto, las tablas *joinrel* y *glist* almacenan las igualdades que permiten verificar las **reglas de consistencia** (TMCD) del tipo $\langle \bigsqcup, = \rangle$ establecidas en el trabajo de S. Hegner y M. A. Rodríguez (2016), y que fueron descritas en el capítulo de Discusión Bibliográfica. Estas relaciones de igualdad tienen la forma mostrada en la ecuación.

$$g_n = \bigsqcup \{g_i : g_i \in glist_n\}$$

Podría decirse que *joinrel* representa al elemento g_n al lado izquierdo de la igualdad, mientras que *glist* agrupa a los elementos g_i al lado derecho de esta.

Tablas de instancia
Estas tablas contienen los datos que describen cada instancia de datos, es decir, corresponden a las bases de datos BD_i de la Figura 1. Por lo tanto, pueden haber múltiples tablas de cada tipo con diferentes datos, pero todas tendrán en común la misma estructura de columnas.

- **candidate (name, partido)**
  Esta tabla almacena los candidatos a la Presidencia de Chile en la segunda vuelta del 2013. Contiene una columna *name* con el nombre del candidato, que es clave primaria; y una columna *partido* con su respectivo partido político

- **votos (name, gtty, eleccion, candidato, numvotos, rounding)**
  Existen múltiples tablas de votos, cada una almacena las votaciones de las elecciones a un nivel distinto de granularidad y redondeo. Los votos se identifican por un gránulo (cuya clave primaria es nombre y granularidad) y su candidato. Además se almacena la elección a la que corresponden, así como un atributo numérico (cantidad de votos) y el redondeo al que se encuentra.

Poblamiento de las tablas e información implícita
El modelo multigranular se levantó de una manera tal que la implementación misma permitió observar algunos aspectos que no eran perceptibles a simple vista en los datos iniciales.
Los datos contenidos en las tablas *gttyorder*, *joinrel* y *glist* no estaban almacenados explícitamente en la base de datos inicial que publicó el Servicio Electoral (SERVEL), por lo que fue necesario deducirlos para completar la implementación del modelo.

- **Implementación de gttyorder**
  El orden parcial existente entre las granularidades fue deducido utilizando la regla que se mencionó anteriormente para establecer que una granularidad subsume a otra:

$$\forall g_1 \in Glty_1 \; \exists g_2 \in Glty_2 \; : \; g_1 \sqsubseteq g_2 \Rightarrow Glty_1 \sqsubseteq Glty_2$$

  Sin embargo no todas las relaciones pertenecientes a este orden parcial fueron almacenadas, puesto que la gran mayoría de ellas podían deducirse por transitividad. Lo que realmente contiene la tabla *gttyorder* es la relación de cobertura entre granularidades.

  La generación de los datos de la tabla *gttyorder* fue realizada utilizando el pseudocódigo mostrado en el algoritmo 1. Estos datos permitieron visualizar con claridad el orden parcial existente entre las granularidades de este dominio. Los resultados fueron representados en un *Diagrama de Hasse*, un grafo donde los nodos representan las granularidades del dominio y los arcos (a, b) representan el hecho de que la granularidad

Algoritmo 1 Generación de datos de gttyorder

procedure GTTYORDERGEN
  for $Gtty_1$ in *gttyorder* **do**
    for $Gtty_2$ in *gttyorder* **do**
      $A \leftarrow \{g_a \in Glty_1\}$
      $B \leftarrow \{g_b \in Glty_1 : covers(g_b, g) \wedge g \in Gtty_2\}$
      if $A \cap B = A$ **then**
        insertar $(Glty_1, Glty_2)$ en *gttyorder*

a tiene una relación de cobertura con b. Dicho diagrama se muestra en la Figura 3. Una propiedad importante de este grafo es que si existe un camino desde un nodo $Glty_1$ hasta un nodo $Glty_2$, esto significa que $Glty_1 \sqsubset Glty_2$. Además, se puede ver que en este orden parcial el elemento máximo es la granularidad País, mientras que el mínimo es la granularidad Mesa Electoral.

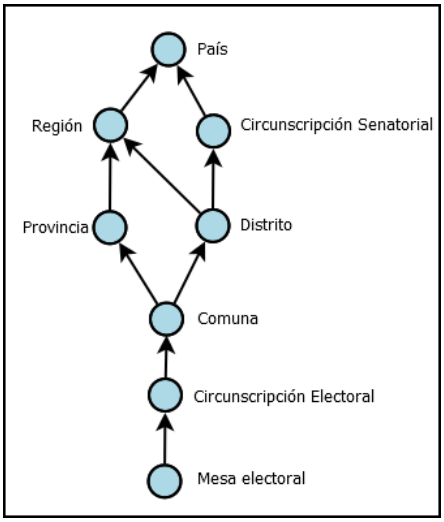

Figura 3: Diagrama de Hasse que representa las relaciones de cobertura y orden existentes en el dominio granular estudiado.

- **Generación de joinrel y glist**
  Las tablas *joinrel* y *glist* también se poblaron utilizando un algoritmo, puesto que la información que contenían no estaba explícita en las bases de datos iniciales.
  Para generar estos datos se hizo uso de las relaciones de cobertura entre gránulos, que sí estaban enunciadas en la base de datos entregada por el SERVEL, además de dos propiedades que poseen todas las divisiones administrativas y electorales de Chile, y que fueron enunciadas previamente en la Sección 3.2.2 "Cumplimiento de las propiedades de join y orden". Estas propiedades son:

- Que todo el territorio nacional está dentro de alguna división administrativa o electoral, sea cuál sea ésta. Cualquier punto del país se encuentra dentro de una Comuna, Distrito o Provincia (Propiedad de *Join*).
- Que la intersección entre dos gránulos de la misma granularidad siempre es vacía. No existe un punto dentro del país que pertenezca a dos comunas, provincias o regiones a la vez (Propiedad de *Disjointness*).

Por lo tanto, siempre podemos asumir que para cualquier gránulo g_1 con granularidad distinta a la mínima (Mesa Electoral), la unión de todos los gránulos g_2 de una misma granularidad $Glty_2$ tales que $g_2 \sqsubset g_1$ es igual a g_1. Como consecuencia de esto, se puede generar el contenido de las tablas *joinrel* y *glist* utilizando un algoritmo.
Vale la pena mencionar que existen casos en que un gránulo g puede descomponerse de más de una manera. Este es el caso de la granularidad Región, cuyos gránulos pueden descomponerse en Provincias o en Distritos. En este tipo de casos la tabla *joinrel* almacena en filas diferentes cada descomposición, de manera que una *id* dirige a la descomposición de la región en provincias y a la descomposición en distritos. Lo mismo ocurre para País, que puede dividirse en regiones o en circunscripciones senatoriales, y en general para cualquier granularidad que tenga más de un antecesor.

Algoritmo 2 Generación de datos de joinrel y glist

procedure JOINREL-GLIST GEN ($granule$, $covers$)
  $i \leftarrow 0$
  $C \leftarrow \{g \in granule : Glty(g) \neq MesaElectoral\}$
  for g in C **do**
    insertar $(i, g, Glty(g))$ en $joinrel$
    $Tmpgtly \leftarrow \{G \in granularity : \exists x \in granule : covers(g, x) \wedge Glty(x) = G\}$
    for t in $Tmpgly$ **do**
      $Gl \leftarrow \{g_0 ingranule : covers(g, x) \wedge Glty(g_0) = t\}$
      for l in Gl **do**
        insertar $(i, l, Glty(l))$ en $glist$
    $i \leftarrow i + 1$

3.3. Desarrollo del Algoritmo de Chequeo de Consistencia

Una vez que se tuvo implementada y poblada la base de datos del dominio granular, así como las bases de datos de instancias, se pasó a la segunda etapa del trabajo, que consistió en el desarrollo del algoritmo propiamente tal que se planteaba al comienzo de este trabajo.

3.3.1. Supuestos previos

A continuación se detallarán los supuestos y condiciones sobre los que se construyó el algoritmo.

- **Esquema de las tablas de datos**
  El algoritmo se diseñó asumiendo que se quiere chequear la consistencia de datos entre dos tablas de datos con el **mismo esquema**, pero que no necesariamente están al mismo nivel de granularidad o redondeados al mismo número.

Tabla 13: Esquema de las tablas de votos

gránulo	granularidad	elección	candidato	numvotos	rounding
g_i	$Glty$	$P2013V2$	c	n	r

- **Esquema de granularidades**
  El algoritmo está desarrollado de manera que entregue el resultado correcto independiente del orden que constituyan las granularidades. Sin embargo, se asume que el orden parcial de granularidades es *fijo*, es decir, que no va a cambiar para las distintas instancias de datos dentro de un mismo dominio granular. También se asume que este esquema es *completo*, o en otras palabras, que contiene toda la información del dominio que es necesaria para realizar las operaciones requeridas por el algoritmo.

  Puesto que el orden de las granularidades es fijo, puede almacenarse y utilizarse una tabla de distancias entre granularidades.

Tabla 14: Distancias entre granularidades

Granularidad 1	Granularidad2	Distancia
$Glty_1$	$Glty_2$	$d(Glty_1,\ Glty_2)$

Se define la distancia entre dos granularidades $Glty_1$ y $Glty_2$ como la cantidad de arcos existentes en el camino más corto de $Glty_1$ a $Glty_2$. Si no existe un camino entre estos nodos, la distancia es infinita.

- **Contenido de la tabla**
  Se asume que todos los valores numéricos de una misma base de datos son enteros positivos, redondeados a un mismo número utilizando la función de redondeo definida previamente. Además se asume que **los gránulos de cada tabla están todos a la misma granularidad**. En las secciones posteriores de este documento se hablará de la **granularidad de una tabla** para referirse a la granularidad a la que se encuentran los datos contenidos en esa tabla.

- **Agregación**
  En el caso de las votaciones la manera de agregar los votos al pasar de una granularidad a otra es la suma. Si bien el algoritmo implementado asume que la operación de agregación de datos es la suma, no debiera ser de mayor complejidad reemplazar esta operación por otra siempre y cuando se cumplan las propiedades de monotonicidad y duplicado-invariancia. Ejemplos de otras operaciones de agregación que cumplen estas propiedades son el operador que entrega el elemento máximo de una tabla o el mínimo.

3.3.2. Funcionamiento del Algoritmo

El algoritmo tiene por objetivo dadas dos tablas de datos, determinar si éstas son consistentes. Para ello, su funcionamiento a grandes rasgos debería ser el siguiente:

1. Llevar a ambas tablas a una granularidad común que permita hacer comparaciones.
2. Llevar a ambas tablas a un nivel de redondeo común.
3. Comparar los datos y entregar los resultados pertinentes.

El primer punto se consigue determinando cuál es la granularidad común a la que se deben llevar los datos, a fin de que la información se encuentre con el mayor nivel de detalle posible. La granularidad que se utiliza es el **supremo** (*lowest upper bound*) entre las granularidades a las que se encuentran los elementos de dos tablas que se comparan, y se calcula utilizando la tabla de distancias que se generó para el esquema.
El supremo de un par de granularidades A y B se entiende intuitivamente como el *menor* elemento que es mayor o igual a ambas granularidades, y es fácilmente visible en el Diagrama de Hasse (ver Figura 4).
En el caso de que dos granularidades sean comparables, el supremo entre ambas es la granularidad mayor de las dos, tal como se muestra en la Figura 5.

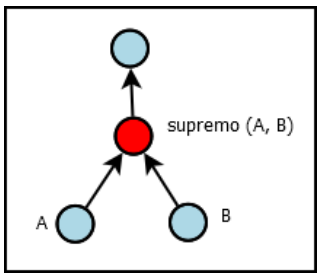

Figura 4: Supremo de dos elementos A y B, marcado en rojo.

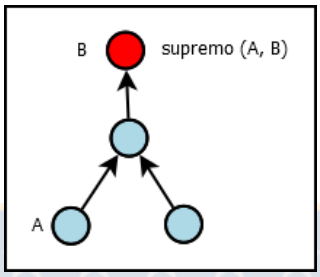

Figura 5: Supremo de dos elementos A y B. En este caso, como $A \sqsubset B$, el supremo es igual a B.

Supremo de dos granularidades
La obtención del supremo de dos granularidades $Glty_1$ y $Glty_2$ se realiza haciendo un ordenamiento por anchura (*Breadth First Search*) de los nodos a los que se puede llegar desde $Glty_1$. El recorrido por anchura se hace utilizando una fila (*Queue*) en la que se van insertando los elementos adyacentes a cada nodo del grafo.
Para cada nodo recorrido *tmp* se consulta si la distancia entre ese nodo y $Glty_2$ es no infinita, pues si es que existe un camino de $Glty_2$ a *tmp* significa que este último es un ancestro del primero, y por lo tanto es su supremo. Análogamente, se pregunta si la distancia de *tmp* a $Glty_2$ es no infinita, y de ser así, significa que $Glty_2$ es el supremo. Todo este proceso se muestra en pseudocódigo en el Algoritmo 3.

En el peor de los casos un algoritmo de búsqueda por anchura recorre todo el grafo de granularidades, y su costo es de $O(|V| + |E|)$, donde $|V|$ es la cantidad de granularidades (nodos) y $|E|$ es la cantidad de arcos del grafo, que corresponde a la cantidad de pares de granularidades con relación de cobertura, y es igual a la cantidad de filas de la tabla *gttyorder*.

Este algoritmo representa una subrutina del chequeo de consistencia y por lo tanto su costo de ejecución se considerará como parte del algoritmo principal.

Algoritmo 3 Búsqueda de supremo de dos granularidades

```
procedure LUP (Glty_1, Glty_2)
    raiz ← PAIS
    Q ← newQueue()
    tmp ← Glty_1
    while tmp ≠ raiz do
        if distancia(Glty_2, tmp) ≠ ∞ then return tmp
        for v in adyacencia(tmp) do
            enqueue(Q, v)
        tmp ← dequeue(Q)
    return raiz
```

Conversión entre granularidades

El algoritmo de chequeo de consistencia utiliza también una subrutina de conversión de granularidades, que lleve los gránulos de una tabla desde la granularidad en que se encuentran en un principio hasta la granularidad que se determinó como **supremo** entre las granularidades de la otra tabla.

La rutina comienza calculando el **camino** en el grafo que va desde la granularidad a la que se encuentra la tabla hasta la granularidad G que se determinó como supremo en el paso anterior. Esto se realiza con el Algoritmo de Dijkstra.

Cuando se tiene el camino a seguir, se van agregando y agrupando los gránulos correspondientes según los datos que se encuentran en las tablas *joinrel* y *glist*, que permiten obtener los gránulos componentes de un gránulo a cierta granularidad.

Este procedimiento se muestra en pseudocódigo en el algoritmo 4. En él se denota la operación de agregación con el símbolo "$\bigoplus$" y es equivalente a la expresión GROUP BY de PL/pgSQL. Las demás operaciones sobre tablas están en notación de álgebra relacional.

Algoritmo 4 Conversión entre granularidades

```
procedure GLTYCONV (Table, Rounding, GltyInicial, GltyFinal)
    temp table ret ← Table
    temp table path ← camino(GltyInicial, GltyFinal)
    while i = fetch(path) ≠ NULL do
        temp table tvotos ← ret ⋈_(ret.name=glist.name, ret.gtty=ret.gtty) glist
        temp table jr ← σ_(gtty=i) joinrel
        temp table agg ← jr ⋈_(jr.id=tvotos.id) tvotos
        agg ← ⨁_gtty agg
        ret ← agg
    return ret
```

El costo de ejecución de esta rutina se calculará a continuación. Se considerará que se tienen n gránulos y $|V|$ granularidades.

- Puesto que el cálculo del camino de granularidades se realiza utilizando una tabla de distancias previamente generada, tiene un costo de ejecución de $O(|V|^2)$ en el peor de los casos.

- El ciclo *while* itera en el peor de los casos $|V|$ veces, cuando hay que recorrer todo el grafo realizando las agregaciones. Para cada iteración los costos de ejecución son los siguientes:

  - Para calcular *tvotos* se realiza una operación *join natural* ("$\bowtie$"). Puesto que este operador se define en base al producto cartesiano de tablas, su complejidad en el peor de los casos es de $O(n^2)$. Sin embargo, la cantidad de elementos de la tabla es $O(n)$.
  - La obtención de *jr* se realiza con una operación de *select* ("σ") que debe recorrer toda la tabla. La tabla *joinrel* en el peor de los casos tiene una cantidad $O(n)$ de elementos, por lo que esta operación también es $O(n)$.
  - La operación *join natural* para calcular *agg* es $O(n^2)$, puesto que se realiza sobre dos tablas con una cantidad de elementos $O(n)$.
  - La operación de agregación realizada sobre la tabla *agg* es sobre una cantidad $O(n)$ de elementos y es de tiempo constante para las agregaciones usuales (suma, promedio, máximo, mínimo). Por lo tanto es $O(n)$.
  - Sobreescribir la tabla *ret* es una operación $O(n)$, puesto que *agg* tiene $O(n)$ elementos.

  Sumando todas las operaciones realizadas dentro del ciclo while y multiplicándolas por la cantidad de iteraciones realizadas tenemos

$$O(|V|) * (O(n^2) + O(n) + O(n^2) + O(n) + O(n)) \;=\; O(|V| * n^2)$$

Finalmente, sumando todas las operaciones que de hacen en la subrutina de conversión entre granularidades, se tiene:

$$O(|V|^2) + O(|V| * n^2) = O(|V| * n^2)$$

3.3.3. Algoritmo

Utilizando las dos subrutinas descritas anteriormente, se puede implementar el algoritmo de chequeo de consistencia.
Para su funcionamiento se toman como entrada las dos tablas de votos a comparar, así como sus granularidades y sus niveles de redondeo, además de un parámetro *Tol* que determina la tolerancia a las diferencias de datos entre ambas tablas.

Se comienza encontrando la granularidad suprema (*lowest upper bound*) entre las dos granularidades a las que se encuentran las tablas de datos. Luego se llevan las dos tablas a la

granularidad suprema (en caso de que no se encuentren inicialmente en ella) utilizando la subrutina de conversión de granularidades que se mostró previamente.
Una vez hecho esto, el segundo paso es llevar las tablas a un nivel de redondeo común. Para ello se busca el mínimo común múltiplo entre el nivel de redondeo de ambas tablas, y luego estas se aproximan a este valor si es que es distinto del nivel de redondel en el que estaban en un principio.
Posteriormente se genera la tabla *comp* realizando una operación *join* entre las dos tablas (que ahora se encuentran en una misma granularidad y redondeo), lo que permite realizar la comparación entre los valores de las columnas de votos. El último paso es el chequeo de la diferencia promedio entre los votos de ambas tablas: Si esta diferencia es mayor (en valor absoluto) que el parámetro *Tol*, se considera que las tablas son inconsistentes. En el caso contrario, las tablas se consideran consistentes. El procedimiento de chequeo de consistencia se muestra en pseudocódigo en el Algoritmo 5.

Complejidad Computacional
La complejidad computacional de este algoritmo se puede determinar usando los cálculos previos que se hicieron para las subrutinas de obtención de supremo y conversión entre granularidades.
Considerando que se tienen n gránulos, $|V|$ granularidades y $|E|$ relaciones de cobertura entre granularidades, se puede decir que:

- La rutina *lup* que se llama tiene un costo de $O(|V| + |E|)$.
- Copiar las tablas de entrada a tablas temporales $votos_1$ y $votos_2$ tiene un costo de $O(n)$.
- Cada llamado a la rutina de conversión entre granularidades tiene un costo de $O(|V| * n^2)$.
- Encontrar el mínimo común múltiplo entre dos números es una operación cuyo tiempo de ejecución es independiente del tamaño de las tablas de datos, y por lo tanto se considerará de tiempo constante, es decir, $O(1)$.
- La operación de redondeo es de tiempo constante, pero se realiza para cada elemento en las tablas de votos, por lo tanto el ciclo iterativo *for* es $O(n)$
- La operación *join* que se hace sobre las tablas $votos_1$ y $votos_2$ está basada en un producto cartesiano sobre dos tablas con una cantidad $O(n)$ de elementos, y por lo tanto es $O(n^2)$.
- El *select* realizado sobre la tabla *uvotos* tiene un costo de $O(n)$
- Encontrar el valor máximo de dif tiene también un costo $O(n)$

Algoritmo 5 Chequeo de consistencia

procedure CONSTCHECK ($Tabla_1$, $Glty_1$, $Rounding_1$, $Tabla_2$, $Glty_2$, $Rounding_2$, Tol
  $supremo \leftarrow lup\ (Glty_1,\ Glty_2)$
  temp table $votos_1 \leftarrow Tabla_1$
  temp table $votos_2 \leftarrow Tabla_2$
  $mcm \leftarrow minimoComunMultiplo\ (Rounding_1,\ Rounding_2)$
  if $mcm \neq Rounding_1$ **then**
    for row in $votos_1$ **do**
      $row.numvotos \leftarrow row.numvotos + Rounding_1/2$
  if $mcm \neq Rounding_2$ **then**
    for row in $votos_2$ **do**
      $row.numvotos \leftarrow row.numvotos + Rounding_2/2$
  if $Glty_1 \neq supremo$ **then**
    $votos_1 \leftarrow gltyConv\ (votos_1,\ Rounding_1,\ Glty_1,\ supremo)$
  if $Glty_2 \neq supremo$ **then**
    $votos_2 \leftarrow gltyConv\ (votos_2,\ Rounding_2,\ Glty_2,\ supremo)$
  if $mcm \neq Rounding_1$ **then**
    for row in $votos_1$ **do**
      $row.numvotos \leftarrow redondear(row.numvotos,\ mcm)$
  if $mcm \neq Rounding_2$ **then**
    for row in $votos_2$ **do**
      $row.numvotos \leftarrow redondear(row.numvotos,\ mcm)$
  temp table $uvotos \leftarrow votos_1 \bowtie_{votos_1.name=votos_2.name,\ votos_1.gtty=votos_2.gtty} votos_2$
  temp table $comp \leftarrow \sigma_{votos_1.name,votos_1.gtty,|votos_1.numvotos-votos_2.numvotos|\ as\ dif}(uvotos)$
  if $avg\ (comp.dif) > Tol$ **then**
    return inconsistente
  else return consistente

Agregando todos los costos de ejecución señalados anteriormente, se tiene:

$$O(|v| + |E|) + O(n) + O(|v| * n^2) + O(1) + O(n) + O(n^2) + O(n) + O(n) = O(|V| * n^2 + |E|)$$

Por lo tanto, el algoritmo de chequeo de consistencia de datos tiene una complejidad de $O(|V| * n^2 + |E|)$

Salida del Algoritmo
La información que devuelve el Algoritmo 5 corresponde a una salida binaria —consistente o inconsistente—. Sin embargo, puede ser útil para el análisis de los resultados entregar más parámetros de salida.
Una manera de devolver más de un elemento en una función PLpgSQL es hacer que la función en cuestión retorne una tabla con una única fila y cuyas columnas devuelvan los parámetros que se necesiten. Para el algoritmo de chequeo de consistencia, se decidió utilizar los parámetros mostrados en la tabla 15.

Tabla 15: Esquema de la tabla de salida entregada por el algoritmo

Consistente	Maxdif	Avgdif
boolean	*integer*	*float*

- **Consistente** es un parámetro de tipo *boolean* cuyo valor es verdadero si las tablas son consistentes, o falso si es que no lo son
- **Maxdif** es de tipo *integer*, y representa la diferencia máxima que se encontró entre los valores numéricos de las tablas de votos. Se calcula como máx($|numvotos_1 - numvotos_2|$)
- **Avgdif** es la diferencia promedio entre los valores numéricos de las tablas de votos. Se calcula como promedio($|numvotos_1 - numvotos_2|$)
- **A**demás se medirá el tiempo de ejecución del código PLpgSQL en milisegundos.

3.3.4. Determinación de un Umbral de Error

Uno de los parámetros de la entrada del algoritmo desarrollado es el valor llamado *tolerancia* o *umbral*, que determina hasta qué punto el algoritmo puede asumir que las inconsistencias entre datos son producto de pérdida de precisión por redondeos o agregaciones. En rigor, el algoritmo entrega el resultado *consistente* si y sólo sí la diferencia promedio entre los valores de las tablas comparadas es menor que la tolerancia fijada. Es por este motivo que es importante saber qué orden de valores para este parámetro permitirán distinguir efectivamente las tablas consistentes de las inconsistentes.

Debido a que el resultado final depende de una comparación con el valor *avgdif*, son los valores que influyen en esta variable los que tendrán una incidencia directa en el resultado del programa. A continuación se nombrarán los factores que afectan el valor de las variables *maxdif avgdif*.

- **Redondeo de las tablas**
  Cuando se realizan las comparaciones de tablas, ambas están redondeadas a un valor R que es el mínimo común múltiplo entre el redondeo inicial de las dos tablas. Por lo tanto, las diferencias entre las tablas siempre serán múltiplos de R. Esto significa que $avgdif$ es un promedio de valores que son múltiplos de R, mientras que *maxdif* es siempre un múltiplo de R.

- **Factor de agregación promedio**
  Otro de los factores que inciden en el valor de $avgdif$ es un parámetro que conoceremos como factor de agregación promedio (fa). Intuitivamente sabemos que las diferencias de datos entre filas en las tablas pueden aparecer o amplificarse al sumar datos redondeados, y que por lo tanto mientras más datos se sumen para pasar de una granularidad a otra, mayor es la potencial diferencia que puede producirse entre las tablas. Esto se puede ilustrar con un ejemplo.

Tabla 16: Tabla de votos de ejemplo

Gránulo	Granularidad	Candidato	numvotos	rounding
Arauco	Provincia	C	v_1	r
Biobío	Provincia	C	v_2	r
Concepción	Provincia	C	v_3	r
Ñuble	Provincia	C	v_4	r

Supongamos que se tiene una base de datos tal como se muestra en la Tabla 16. Si se quisiera convertir esta tabla a la granularidad Región con redondeo $R > r$, el algoritmo suma a cada número de votos un valor igual a la mitad del redondeo r antes de sumar los valores. Por lo tanto, los votos de cada provincia serán iguales a $v_i + r/2$. Luego se suman y redondean estos valores, lo que entregaría un resultado que se muestra en la Tabla 17.

Tabla 17: Conversión de la Tabla 16 a Región y redondeo R

Gránulo	Granularidad	Candidato	numvotos	rounding
Biobío	Región	C	$\sum(v_i + r/2)$	R

Puede observarse que existe una relación entre el valor de *numvotos* y la cantidad de filas que se agregan, puesto que a mayor cantidad de filas mayor será el número por el que esté multiplicado r. A partir de esto, definiremos el factor de agregación promedio ($fa(Glty_1,\ Glty_2)$) como la cantidad promedio de filas que se agregan para pasar de una granularidad $Glty_1$ a una granularidad $Glty_2$. Numéricamente, esto puede expresarse como la cantidad de filas de la granularidad $Glty_2$ dividida por la cantidad de filas de granularidad $Glty_1$. En la Tabla 18 se muestran los valores que toma esta variable para todas las conversiones posibles desde la granularidad Circunscripción Electoral.

Tabla 18: Factor de agregación promedio para conversiones desde Circunscripción Electoral

Tabla 1	**Tabla 2**	$fa(Glty_1,\ Glty_2)$
Circunscripción Electoral	Comuna	1,65
Circunscripción Electoral	Distrito	9,50
Circunscripción Electoral	Provincia	10,55
Circunscripción Electoral	Circunscripción Senatorial	30,00
Circunscripción Electoral	Región	38,00
Circunscripción Electoral	País	570,00

La efectividad de estos dos parámetros a la hora de determinar un umbral efectivo para el algoritmo será estudiada en el capítulo Experimentos y Resultados.

4. Experimentos y Resultados

La sección presente detalla el proceso de validación del algoritmo y los resultados que se obtuvieron de él. Las pruebas fueron realizadas con dos objetivos, el primero de ellos fue mostrar que el algoritmo entrega resultados correctos en distintos casos, mientras que el segundo fue determinar si puede estimarse un umbral de error efectivo a partir de los parámetros estudiados.

4.1. Detalles de la implementación

Se realizaron pruebas de tiempo de ejecución correspondientes al algoritmo de Chequeo de Consistencia, presentado en el Algoritmo 5 de este documento.

La base de datos fue implementada en PostgreSQL 9.5.10, en el servidor *isci.inf.udec.cl*, mientras que el algoritmo, así como todas sus subrutinas, fueron escritas en PL/PgSQL 9.5.

Si bien inicialmente se implementaron los algoritmos mostrados en C/C++ en conjunto con ODBC, finalmente se optó por PL/pgSQL por sobre lenguajes de tipo procedural porque, a diferencia de ellos, PL/PgSQL está diseñado para trabajar con bases de datos, y ofrece tiempos de ejecución menores y un uso de memoria reducido para trabajar con este tipo de datos.

4.2. Creación de datos de prueba

Para realizar el testeo del algoritmo fue necesario generar datos de entrada con distintos niveles de granularidad, a diferentes valores de redondeo y tanto tablas consistentes e inconsistentes entre sí.
Inicialmente se contaba únicamente con los datos entregados por el SERVEL adaptados al esquema de datos multigranular, por lo que fue necesario desarrollar un programa que permitiera generar datos nuevos a partir de esta tabla base.
El Algoritmo 6 muestra en pseudocódigo el procedimiento de generación de tablas de votos a una determinada granularidad, con cierto nivel de redondeo y con un parámetro *error* que permite generar datos inconsistentes con los datos de base.

Algoritmo 6 Generación de tabla de votos

procedure GENVOTOS($Glty$, $Rounding$, $Error$)
  temp table $ret \leftarrow grcl.votos$
  temp table $path \leftarrow camino('MESA\ ELECTORAL', Glty)$
  while $i = fetch(path) \neq NULL$ **do**
    temp table $tvotos \leftarrow ret \bowtie_{(ret.name=glist.name,\ ret.gtty=ret.gtty)} glist$
    temp table $jr \leftarrow \sigma_{gtty=i} joinrel$
    temp table $agg \leftarrow jr \bowtie_{jr.id=tvotos.id} tvotos$
    $agg \leftarrow \bigoplus_{gtty} agg$
    $ret \leftarrow agg$
  if $Error \neq 0$ **then**
    for row in ret **do**
      $row.numvotos \leftarrow row.numvotos + Error$
  for row in ret **do**
    $row.numvotos \leftarrow aproxnum(row.numvotos,\ Rounding)$
    $row.rounding \leftarrow Rounding$
  return ret

El algoritmo está desarrollado a partir de la subrutina de conversión entre granularidades con algunas modificaciones, pues se busca llevar la tabla de votos base (*grcl.votos*) a la granularidad requerida. Posteriormente, si el parámetro error es distinto de cero, se suma a la cantidad de votos un valor igual al parámetro error. De esta manera se generan datos que son distintos a los votos que tienen las tablas base. Finalmente, se redondean los valores al número especificado.

4.3. Datos generados

Utilizando el algoritmo expuesto anteriormente se generaron tablas de votos a distintas granularidades, con distintos redondeos y con datos tanto consistentes como inconsistentes entre sí. En la Tabla 19 se muestran algunas de las tablas de votos que se generaron con el propósito de hacer pruebas sobre el algoritmo desarrollado.

Tabla 19: Ejemplos de tablas de entrada generadas para los experimentos.

Nombre	**Granularidad**	**Redondeo**	**Error**
votos (base)	Mesa Electoral	1	0
votoscie	Circunscripción Electoral	10	0
votoscie_i1	Circunscripción Electoral	10	1
votoscie_i5	Circunscripción Electoral	10	5
votoscom	Comuna	25	0
votosprov	Provincia	100	0
votosprov_i1	Provincia	100	1
votosprov_i10	Provincia	100	10
votosregion	Región	500	0
votosregion_i100	Región	500	100
votosdist	Distrito	50	0
votoscis10	Circunscripción Senatorial	10	0
votoscis10_i1	Circunscripción Senatorial	10	1
votoscis1000	Circunscripción Senatorial	1000	0
votospais	País	1	0
votospais1000	País	1000	0

Estas tablas serán usadas en pares como entrada para el algoritmo, con el objetivo de estudiar cómo varían los resultados obtenidos para distintos casos.

4.4. Pruebas de Tiempo de Ejecución

El objetivo de los experimentos realizados fue determinar cómo cambian los tiempos de ejecución del algoritmo de Chequeo de Consistencia (Algoritmo 5) según el valor de distintos parámetros de la entrada, comparando los resultados obtenidos de experimentos con los resultados de estudiar el costo analíticamente.

- **La distancia entre las granularidades de las tablas:** Se compararon casos de tablas de granularidades adyacentes y granularidades con distancias mayores. Este valor corresponde al parámetro $|E|$ del costo de ejecución analítico del algoritmo.
- **La cantidad de gránulos de las tablas:** La cantidad de gránulos que existen en las respectivas granularidades de las tablas. Por ejemplo, hay muchas más comunas que regiones. En el costo de ejecución analítico del algoritmo, este parámetro equivale al valor n.

4.4.1. Distancia entre granularidades

Para realizar las pruebas de distancia, se estudió la salida del algoritmo para todas las distancias posibles entre granularidades, para el grafo de divisiones políticas y administrativas de Chile. Como referencia, en la Tabla 25 se muestra la cantidad de filas asociadas a la tabla de cada granularidad.

Tabla 20: **Distancia = 1**

Tabla 1	**Filas 1**	**Tabla 2**	**Filas 2**	**Tiempo Ejecución**
Mesa Electoral	165396	Circunscripción Electoral	2280	1954,613 ms
Circunscripción Electoral	2280	Comuna	1380	232,962 ms
Comuna	1380	Provincia	216	136,325 ms
Provincia	216	Región	60	142,513 ms
Región	60	País	4	121,165 ms
Comuna	1380	Distrito	240	114,447 ms
Distrito	240	Región	60	125,857 ms
Distrito	240	Circunscripción Senatorial	76	129,589 ms
Circunscripción Senatorial	76	País	4	217,897 ms
			Promedio	**379,44 ms**

Tabla 21: **Distancia = 2**

Tabla 1	**Filas 1**	**Tabla 2**	**Filas 2**	**Tiempo Ejecución**
Mesa Electoral	165396	Comuna	1380	2032,996 ms
Circunscripción Electoral	2280	Provincia	216	249,989 ms
Circunscripción Electoral	2280	Distrito	240	283,328 ms
Comuna	1380	Región	60	107,684 ms
Comuna	1380	Circunscripción Senatorial	76	153,420 ms
Provincia	216	País	4	203,601 ms
Distrito	240	País	4	226,847 ms
			Promedio	**465,409 ms**

Tabla 22: **Distancia = 3**

Tabla 1	**Filas 1**	**Tabla 2**	**Filas 2**	**Tiempo Ejecución**
Mesa Electoral	165396	Provincia	216	1958,350 ms
Mesa Electoral	165396	Distrito	240	2009,728 ms
Circunscripción Electoral	2280	Circunscripción Senatorial	76	227,726 ms
Circunscripción Electoral	2280	Región	60	273,875 ms
Comuna	1380	País	4	240,746 ms
			Promedio	**942,085 ms**

Tabla 23: **Distancia = 4**

Tabla 1	**Filas 1**	**Tabla 2**	**Filas 2**	**Tiempo Ejecución**
Mesa Electoral	165396	Circunscripción Senatorial	76	2067,138 ms
Mesa Electoral	165396	Región	60	2078,146 ms
Circunscripción Electoral	2280	País	4	183,345 ms
			Promedio	**1442,876 ms**

Tabla 24: **Distancia = 5**

Tabla 1	**Filas 1**	**Tabla 2**	**Filas 2**	**Tiempo Ejecución**
Mesa Electoral	165396	País	4	2194,390 ms
			Promedio	**2194,390 ms**

La principal conclusión que se puede obtener a partir de los resultados, mostrados en las Tablas 20 a 24, es que el tiempo de ejecución crece a medida que aumenta la distancia entre granularidades (Figura 6). Esto tiene sentido, ya que cada unidad de distancia representa una iteración adicional en la etapa de conversión de granularidades de las tablas.

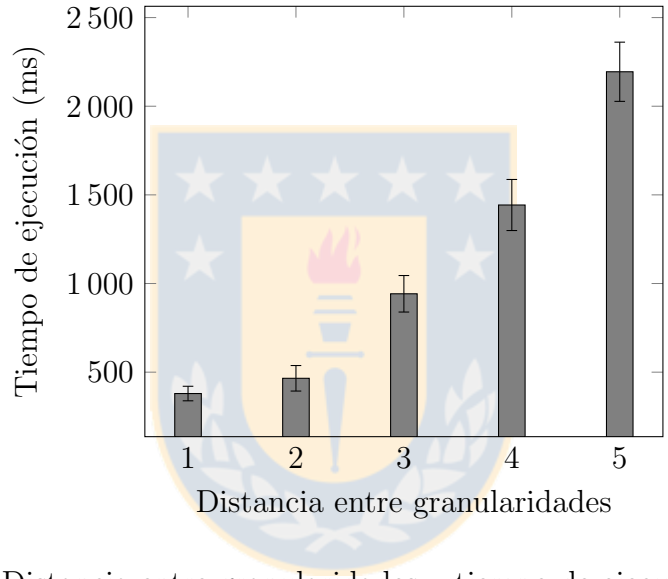

Figura 6: Distancia entre granularidades y tiempo de ejecución (ms)

Es importante observar que en este dominio en particular no se puede hacer variar la distancia $|E|$ entre granularidades sin hacer variar al mismo tiempo la cantidad de gránulos n, puesto que cada granularidad tiene cantidades de gránulos diferentes. Esto explica el hecho de que el crecimiento del tiempo de ejecución en la Figura 6 no es lineal, sino que más bien corresponde a $O(n^2)$.

4.4.2. Cantidad de Gránulos por Granularidad

A modo de continuación de lo anterior, es preciso comparar el comportamiento del algoritmo en relación al número de gránulos de las tablas ingresadas.
Se puede afirmar que mientras más gránulos en total tengan las tablas a comparar, mayor será el tiempo de ejecución. Esto se debe a que hay más operaciones de suma, redondeo y resta que son realizadas. En los resultados de distancia entre granularidades, puede observarse que los chequeos de consistencia con la tabla Mesa Electoral tienen tiempos de ejecución mucho más grandes que los de tablas con menos gránulos como Región o Distrito.

En la Tabla 25 se muestran la cantidad de filas por tabla asociada a cada granularidad. Cabe recordar que la cantidad de filas es igual a cuatro veces la cantidad de gránulos de la granularidad indicada, puesto que se guardan en distintas filas los votos por candidato (dos candidatos más votos blancos y nulos).

Tabla 25: Número de filas por granularidad

Granularidad	**Filas** (n)
Mesa Electoral	165396
Circunscripción Electoral	2280
Comuna	1380
Provincia	216
Región	60
Distrito	240
Circunscripción Senatorial	76
País	4

Para estudiar cómo se relaciona el número de filas n en una tabla con los tiempos de ejecución, se fijó la distancia entre granularidades con el fin de que el único parámetro que varíe sea la cantidad de filas. En la Tabla 26 se muestran los tiempos de ejecución promedio para consultas en las que se compara con cada granularidad, para consultas donde la distancia es igual a 1.

Tabla 26: Tiempo de ejecución promedio por granularidad, para consultas con distancia 1.

Granularidad	**Filas**	**Tiempo de Ejecución (ms)**
Mesa Electoral	165396	1953,613 ms
Circunscripción Electoral	2280	1093,788 ms
Comuna	1380	161,245 ms
Provincia	216	139,419 ms
Región	60	129,845 ms
Distrito	240	123,298 ms
Circunscripción Senatorial	76	173,743 ms
País	4	217,897 ms

En general puede observarse que a medida que crece la cantidad de filas de la tabla, aumenta también el tiempo de ejecución de los chequeos de consistencia asociados a esa tabla. Si bien existen excepciones, estas pueden deberse a propiedades del grafo de granularidades en sí, como el factor de agregación promedio, o a la manera en que PostgreSQL resuelve las consultas.

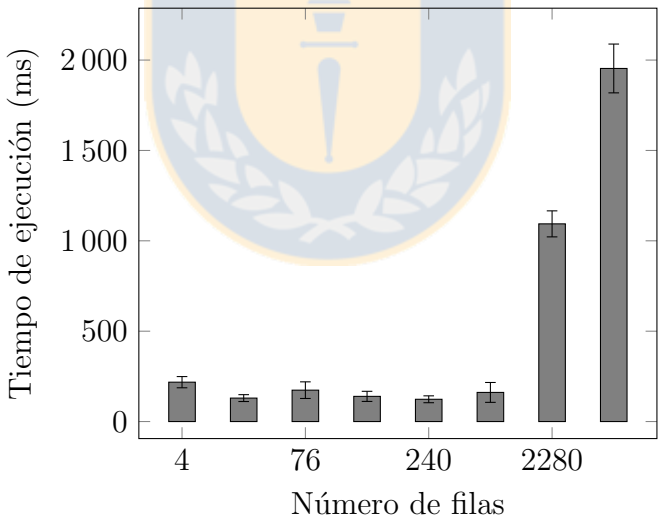

Figura 7: Número de filas y tiempo de ejecución (ms)

4.4.3. Conclusiones de los experimentos de tiempo de ejecución

El comportamiento general del algoritmo sigue los supuestos que se habían hecho antes de realizar los experimentos. Estos eran que el tiempo de ejecución crece con la distancia entre granularidades y con el número de filas. Además, pudo observarse que este crecimiento es similar al que se había determinado analíticamente.

4.5. Pruebas de Umbral de Error

Las pruebas realizadas en esta sección fueron hechas con el objetivo de determinar si se cumplen realmente las relaciones entre parámetros establecidas analíticamente en la Sección 3.3.4 "Determinación de un Umbral de Error". Esto debería servir como guía para establecer un umbral de error (Tol) efectivo para verificar la consistencia de datos.
Las pruebas fueron realizadas con una metodología similar a las pruebas de tiempo de ejecución, donde se hace variar un parámetro y se fijan todos los demás para comparar los resultados obtenidos.

4.5.1. Nivel de Redondeo

Para las pruebas de redondeo, se compararon tablas a distintos niveles de redondeo pero con una misma granularidad con una tabla de Circunscripción Electoral redondeada a 50. Ambas tablas tienen datos correctos, que el algoritmo debiera determinar que son consistentes entre sí. Los resultados obtenidos se muestran en la tabla 27.

Tabla 27: Valor de $avgdif$ para distintos niveles de redondeo

Granularidad	**Redondeo**	**avgdif**
Región	1	95,00
Región	10	100,00
Región	100	936,67
Región	500	900,00
Región	1000	950,00
Región	2000	1100,00
Región	5000	750,33

Los resultados muestran que *avgdif* se mantiene sin mayores variaciones para diferentes niveles de redondeo. Si bien *avgdif* es un promedio de números que son múltiplos de R, para redondeos grandes la mayoría de estos valores son cero.

4.6. Pruebas de Factor de Agregación Promedio

La relación entre el factor de agregación promedio fa y el valor $avgdif$ fue estudiada observando su comportamiento para distintos valores de fa. Los demás parámetros se mantuvieron constantes. Se comparó la tabla de Mesa Electoral con redondeo 10 con distintas tablas, todas ellas con un redondeo de 25.

Tabla 28: Valor de $avgdif$ para distintos niveles de factores de agregación promedio

Granularidad	***fa(i, j)***	**avgdif**	α
Circunscripción Electoral	72,54	340,66	4,70
Comuna	119,85	566,93	4,73
Distrito	689,15	3313,54	4,80
Provincia	765,72	3680,56	4,81
Circunscripción Senatorial	2176,26	10483,60	4,82
Región	2756,60	13284,20	4,82
País	41349,00	199398,00	4,82

Los resultados obtenidos se muestran en la Tabla 28. Puede verse que a medida que crece el factor de agregación promedio fa entre granularidades aumenta también el valor de $avgdif$, tal como se había inferido antes de hacer los experimentos. Sin embargo, es interesante observar que la razón entre $avgdif$ y $fa(i,\ j)$ no experimenta mayores variaciones en todos los casos. Este valor, que llamaremos α, puede ser de importancia para saber estimar un umbral de error correctamente.

El gráfico de la Figura 8 muestra que el valor de $avgdif$ es directamente proporcional al factor de agregación promedio fa, y que su crecimiento es muy similar al de una función lineal.

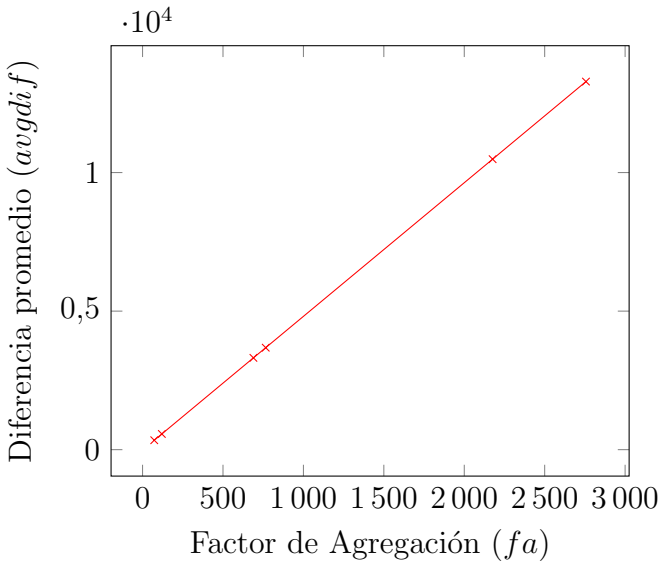

Figura 8: Factor de agregación promedio (fa) y diferencia promedio de datos ($avgdif$)

Sin embargo, todas las tablas mostradas hasta el momento son consistentes. Es necesario contrastar estos datos con tablas en las que existen inconsistencias de datos.

4.7. Pruebas de Chequeo de Consistencia

Finalmente queda estudiar el comportamiento del algoritmo para lo que fue desarrollado inicialmente: Distinguir tablas de datos consistentes entre sí de las que son inconsistentes.

Las diferencias de datos tienen efectos similares en los resultados sean éstas positivas o negativas, puesto que tanto *maxdif* como *avgdif* se calculan utilizando el valor absoluto de la diferencia entre ambas tablas. Sin embargo, si en una tabla que se convierte en otra granularidad se agregan los datos, es posible que las diferencias se anulen entre sí (si hay filas con más votos y otras con menos votos) o bien que éstas desaparezcan en el proceso de redondeo de los datos. En estos casos el algoritmo no necesariamente podría detectar las inconsistencias de datos.

Los experimentos de esta sección se realizaron comparando una tabla de granularidad Región con redondeo 100, con una tabla de granularidad Comuna y redondeo 50. Para las tablas de Comuna hizo variar el parámetro *error*, cuyo funcionamiento se describió en la sección Datos Generados de este capítulo. Cabe mencionar que el factor de agregación promedio de Comuna a Región es 23.

Tabla 29: Valor de $avgdif$ para distintos niveles de error, con $fa(Comuna,\ Region) = 23$

Granularidad	**Error**	**avgdif**	α
Comuna	0	601,67	26,16
Comuna	1	615,00	26,74
Comuna	2	631,67	27,46
Comuna	5	703,33	30,58
Comuna	10	838,33	37,32
Comuna	25	1220,00	53,04
Comuna	50	1751,67	76,16

Los resultados de la Tabla 29 muestran que efectivamente el valor de *avgdif* crece cuando hay inconsistencias, tal como se había pensado. Además puede observarse que el valor de α se comporta de la misma manera, creciendo a medida que se aumenta el error en las tablas.

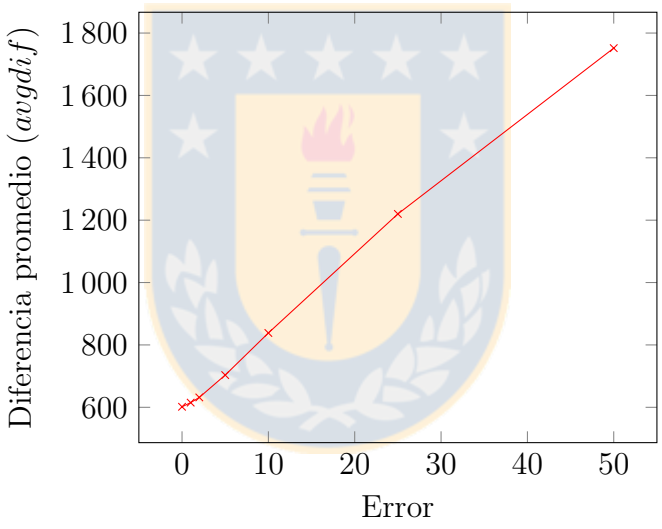

Figura 9: Factor de agregación promedio (fa) y diferencia promedio de datos ($avgdif$)

En el gráfico de la Figura 9 puede verse que la relación existente entre $avgdif$ y el error tiene un comportamiento semejante al de una función lineal.

4.8. Conclusiones de los Experimentos

El comportamiento general del algoritmo sigue los supuestos que se habían hecho antes de realizar los experimentos. Se observó que el tiempo de ejecución crece con la distancia entre granularidades, y que el nivel de redondeo y el factor de agregación promedio son los principales factores que inciden en los valores de *avgdif*.
Si bien se pudo mostrar lo anterior, es difícil establecer una línea clara a seguir para determinar un valor ideal del parámetro *Tol*, que permita detectar efectivamente las inconsistencias de datos. Puesto que se trabaja con números redondeados, se debe aceptar que existirá siempre para estos casos una pérdida de precisión y que no se pueden detectar las inconsistencias en todos los casos.

5. Conclusiones

Se implementó un esquema de datos multigranular así como algoritmos que permiten generar este esquema para distintas bases de datos multigranulares. Posteriormente se adaptó a este esquema la base de datos de la Segunda Vuelta de las Elecciones Presidenciales de Chile 2013, entregada por el SERVEL.
Además se desarrolló un algoritmo que permite realizar el chequeo de consistencia entre tablas de datos a distintas granularidades, verificando el cumplimiento de las restricciones de integridad correspondientes al primer caso ($\langle \sqcup, = \rangle$) de las TMCD planteadas en*A Model for Multigranular Data and its Integrity (S. Hegner & M. A. Rodrguez, 2016)*.
Finalmente, se realizaron pruebas sobre dicho algoritmo con el fin de determinar las relaciones existentes entre los resultados entregados y las características de cada tabla, y se encontró una relación entre el factor de agregación promedio y las diferencias encontradas en los datos de las tablas, aún en tablas sin inconsistencias de datos.

Quedan por desarrollar varios temas:

- Implementar la verificación de consistencia para los otros casos de las TMCD, con distintos valores para los parámetros β y η.
- Expandir el algoritmo para trabajar con múltiples granularidades en una misma tabla. Esto a su vez puede dividirse en dos casos: gránulos que no se superponen entre sí y gránulos superpuestos (intersección no necesariamente vacía).
- Trabajar con distintos niveles de redondeo en una misma tabla. Esto involucraría encontrar un redondeo común entre n valores.
- Implementar el modelo de datos multigranular y el algoritmo de chequeo de consistencia en otras bases de datos, quizás también utilizando operadores de agregación distintos como el elemento máximo o mínimo.

6. Bibliografía

[1] S. Hegner & M. A. Rodríguez. *A Model for Multigranular Data and its Integrity*, 2016.

[2] M. A. Rodríguez & L. Bravo. *Multi-Granular Schemas for Data Integration*, 2012

[3] E. Bertino, E. Camossi & M. Bertolotto. *Multi-Granular Spatio-Temporal Object Models: Concepts and Research Directions*, 2010.

[4] Simovici, Dan A. & Djeraba, Chabane. *Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics*, Ch. 4: Partially Ordered Sets, 2008.